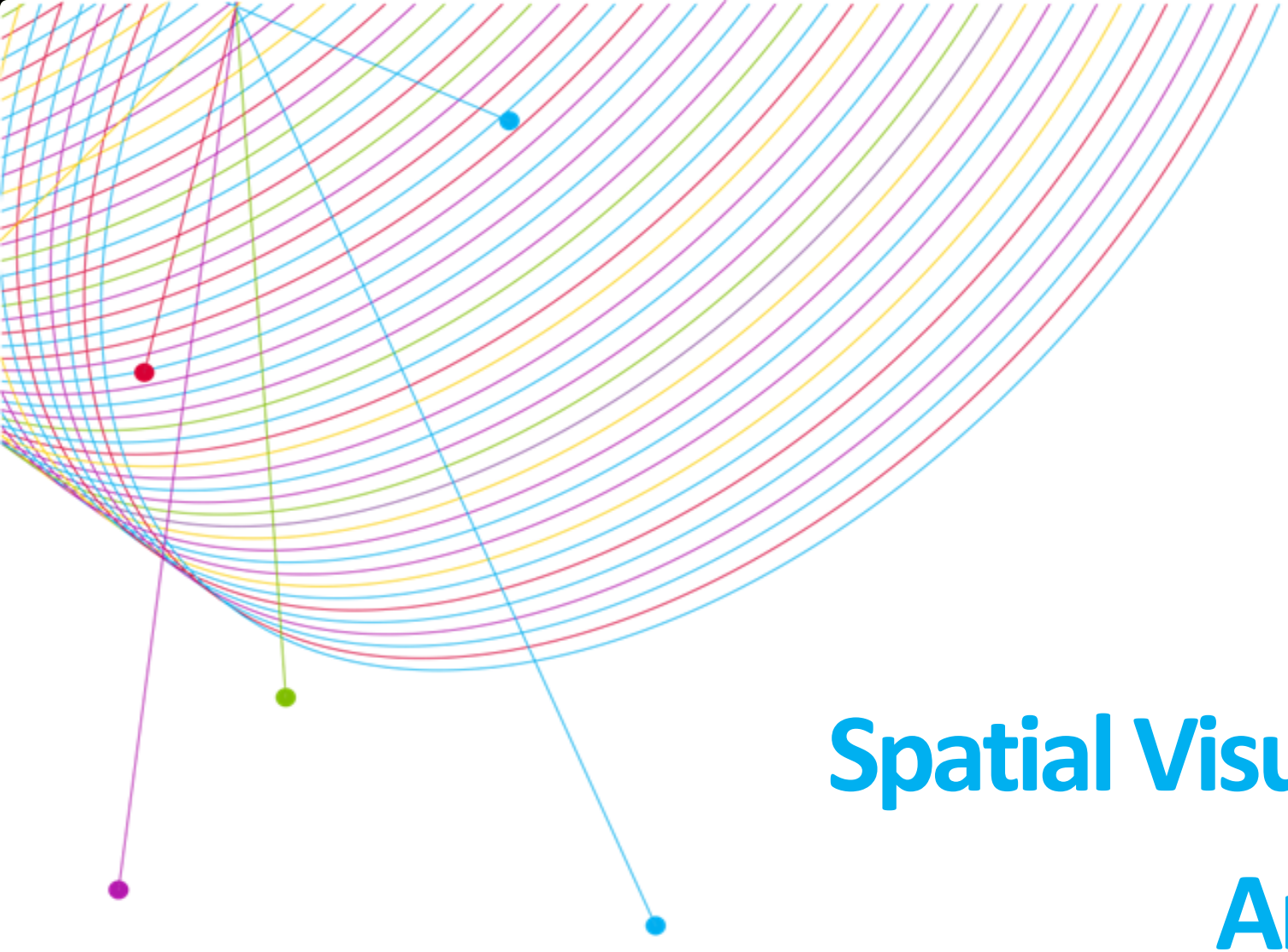
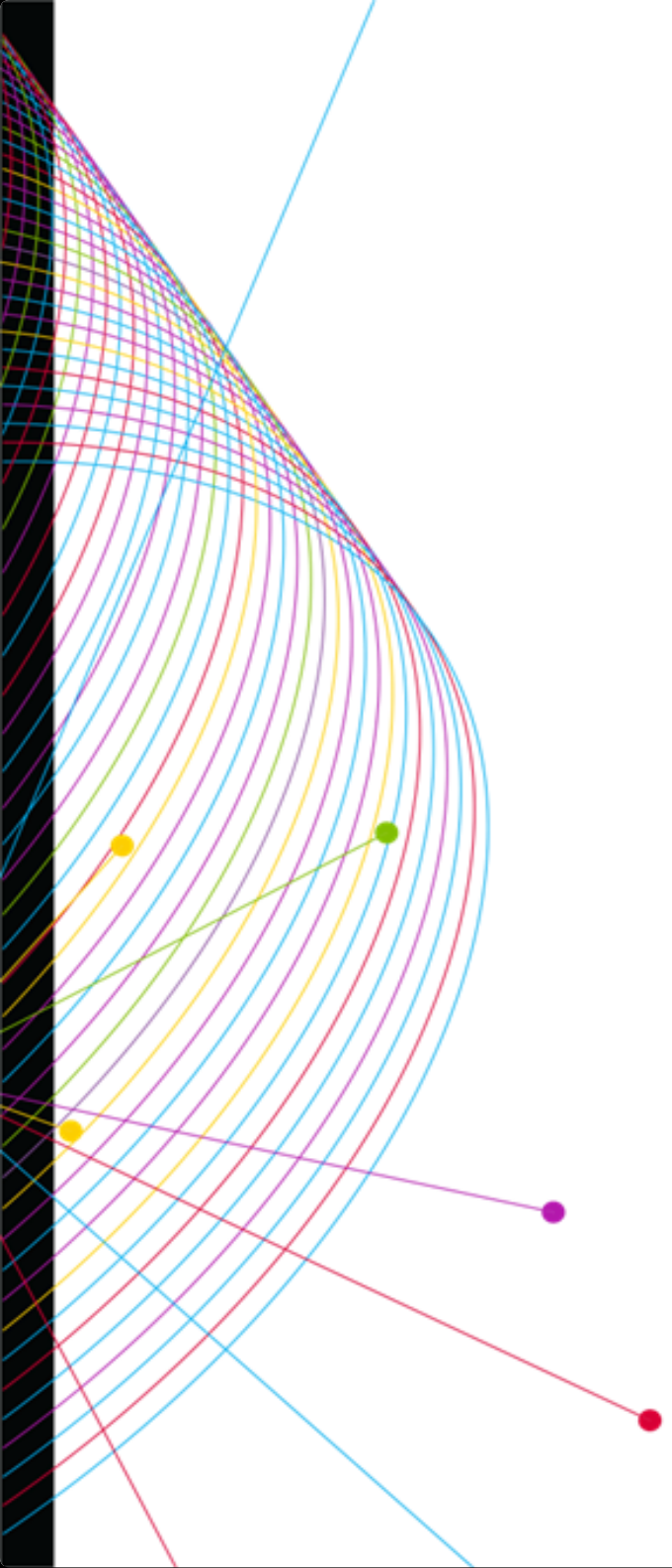


nielsen

Spatial Visualisation and Analysis with R

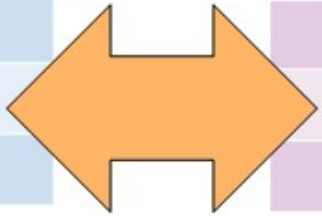
Chong Jie Lim, Timothy Banks and Xiaoyu Lin
Data Science Singapore Innovation, The Nielsen Company (Singapore)



Matching Two Lists of Stores

DATA FIELDS

Store Name	Address Fields



Store Name	Address Fields

An obvious solution:
Fuzzy string-matching

CHALLENGES

GOOGLE MAPS API

Geocoding Module (over HTTP)

- Input: address string
- Return: Address object (JSON/XML)
- standardized address
- geo-coordinates
- quality of address match
- other address attributes like city, locality, etc.

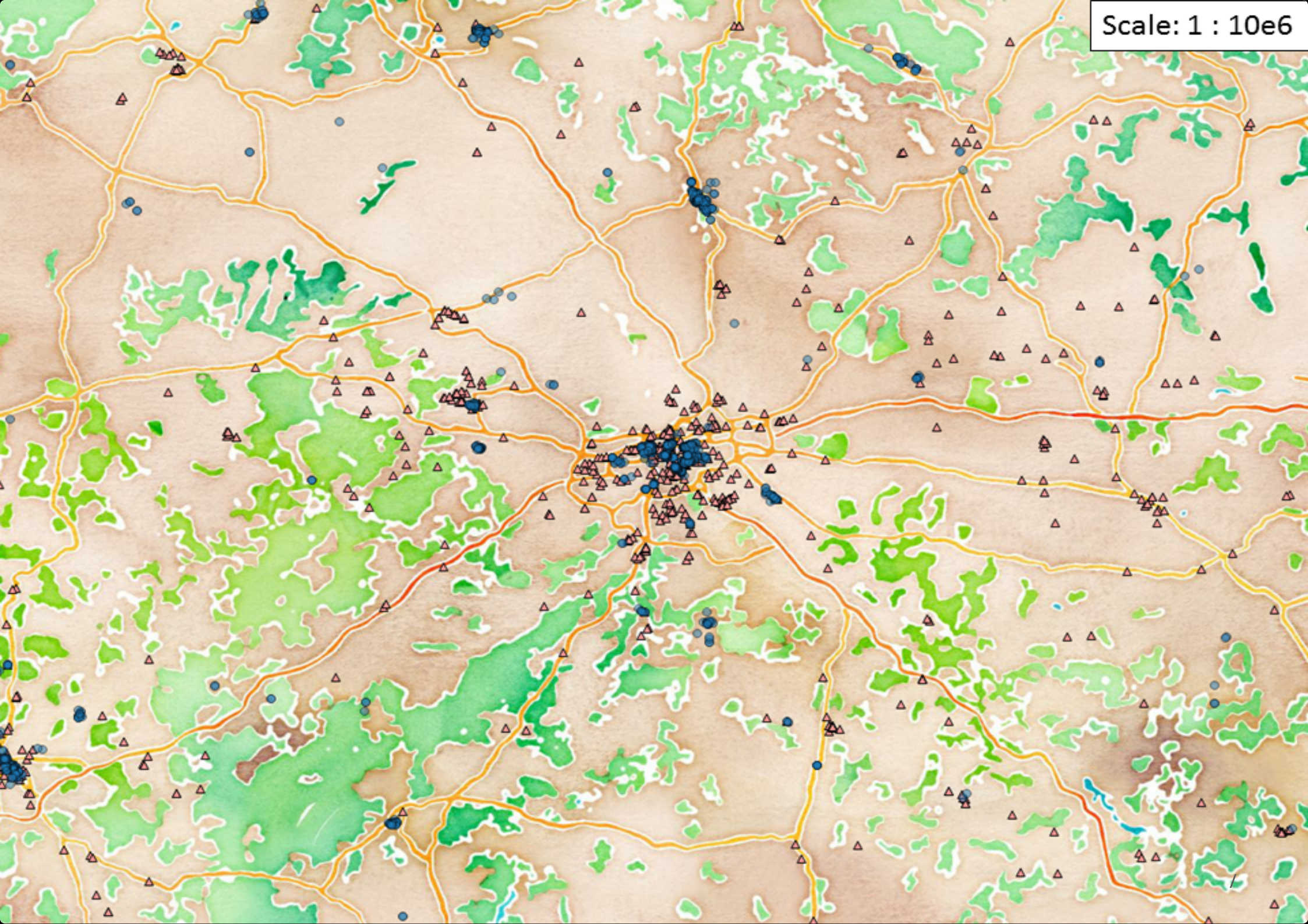
```
{
   "results" : [
      {
         "address_components" : [
            {
               "long_name" : "1600",
               "short_name" : "1600",
               "types" : [ "street_number" ]
            },
            {
               "long_name" : "Amphitheatre Pkwy",
               "short_name" : "Amphitheatre Pkwy",
               "types" : [ "route" ]
            },
            {
               "long_name" : "Mountain View",
               "short_name" : "Mountain View",
               "types" : [ "locality", "political" ]
            },
            {
               "long_name" : "Santa Clara County",
               "short_name" : "Santa Clara County",
               "types" : [ "administrative_area_level_2", "political" ]
            },
            {
               "long_name" : "California",
               "short_name" : "CA",
               "types" : [ "administrative_area_level_1", "political"
            },
            {
               "long_name" : "United States",
               "short_name" : "US",
               "types" : [ "country", "political" ]
            },
            {
               "long_name" : "94043",
               "short_name" : "94043",
               "types" : [ "postal_code" ]
            }
         ],
         "formatted_address" : "1600 Amphitheatre Parkway, Mountain View, CA 94043,
         "geometry" : {
            "location" : {
               "lat" : 37.4224764,
               "lng" : -122.0842499
            },
            "location_type" : "ROOFTOP",
            "viewport" : {
               "northeast" : {
                  "lat" : 37.4238253802915,
                  "lng" : -122.0829009197085
               },
               "southwest" : {
                  "lat" : 37.4211274197085,
                  "lng" : -122.0855988802915
               }
            }
         },
         "place_id" : "ChIJ2eUgeAK6j4ARbn5u_wAGqWA",
         "types" : [ "street_address" ]
      }
   ],
   "status" : "OK"
}
```

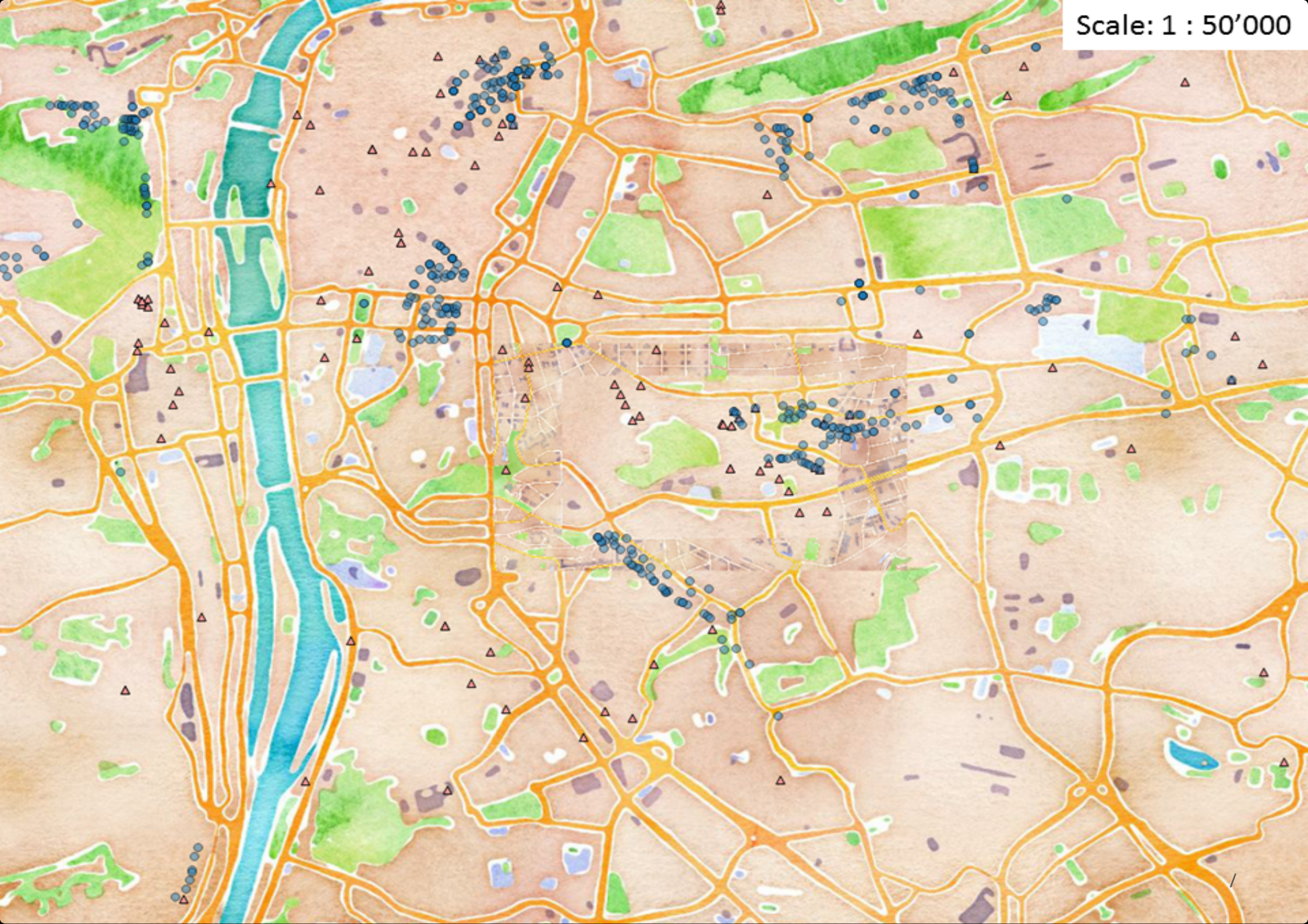
MATCHING METHODOLOGY

Combination of

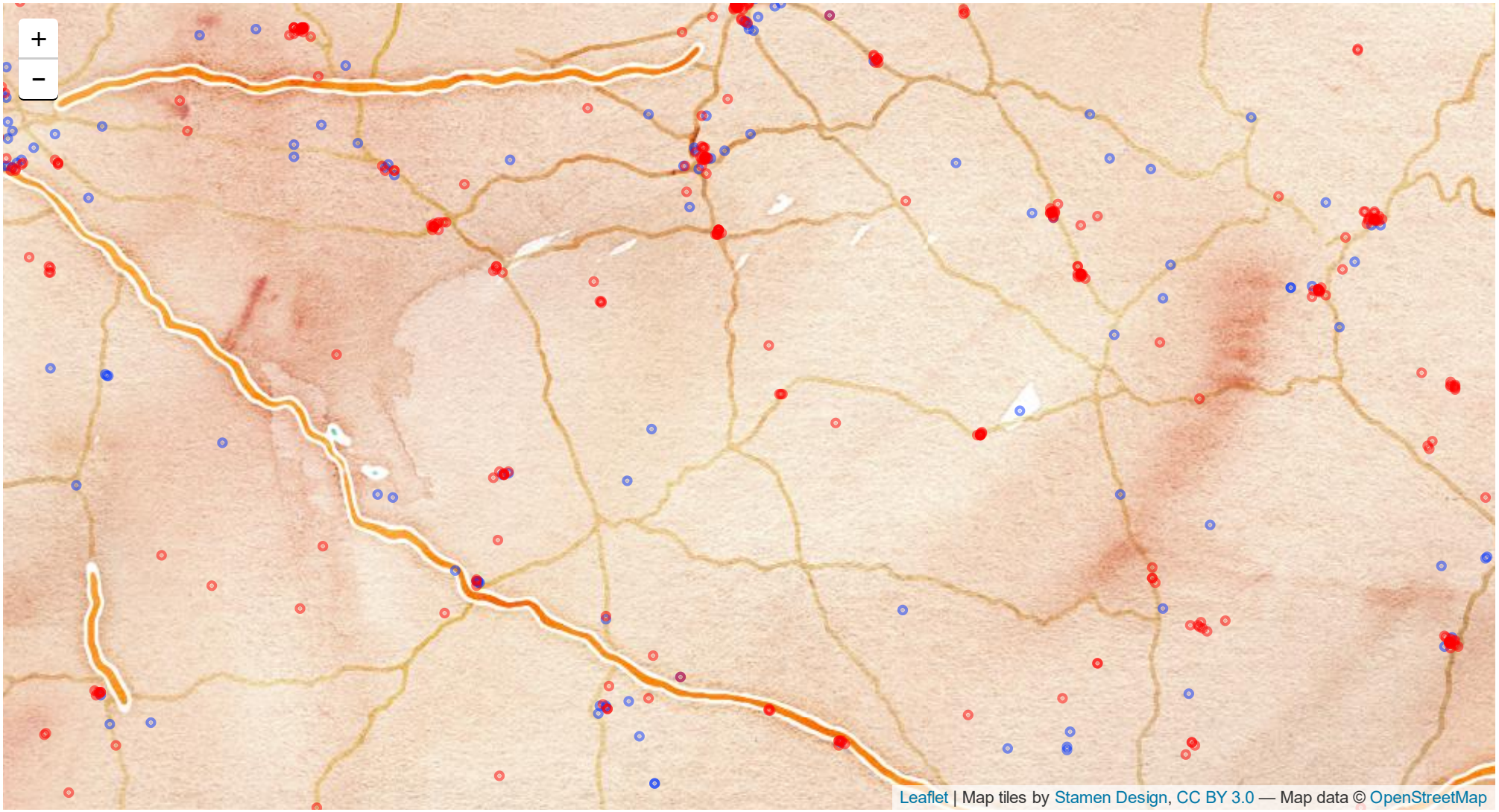
- Geographical Fencing
- Fuzzy String Matching

However, we obtained poor match rates!


Scale: 1 : 10e6


Scale: 1 : 50’000

INTERACTIVE FORMAT USING LEAFLET



A LITTLE ABOUT LEAFLET FOR R

- Based on JavaScript
- Layer-based syntax
- Works with `magrittr` pipe symbol `%>%`
- Works with lat/long, sp polygon objects, external map tiles
- Produces interactive html-based plots

LEAFLET BASICS

- Initialize with `leaflet` function, either with or without data input
- Add map tiles using `addTiles` or `addProviderTiles` function to create the "background" if necessary
- Add objects like markers, circles, polygons, lines, popups, etc.

LEAFLET "HELLO WORLD"

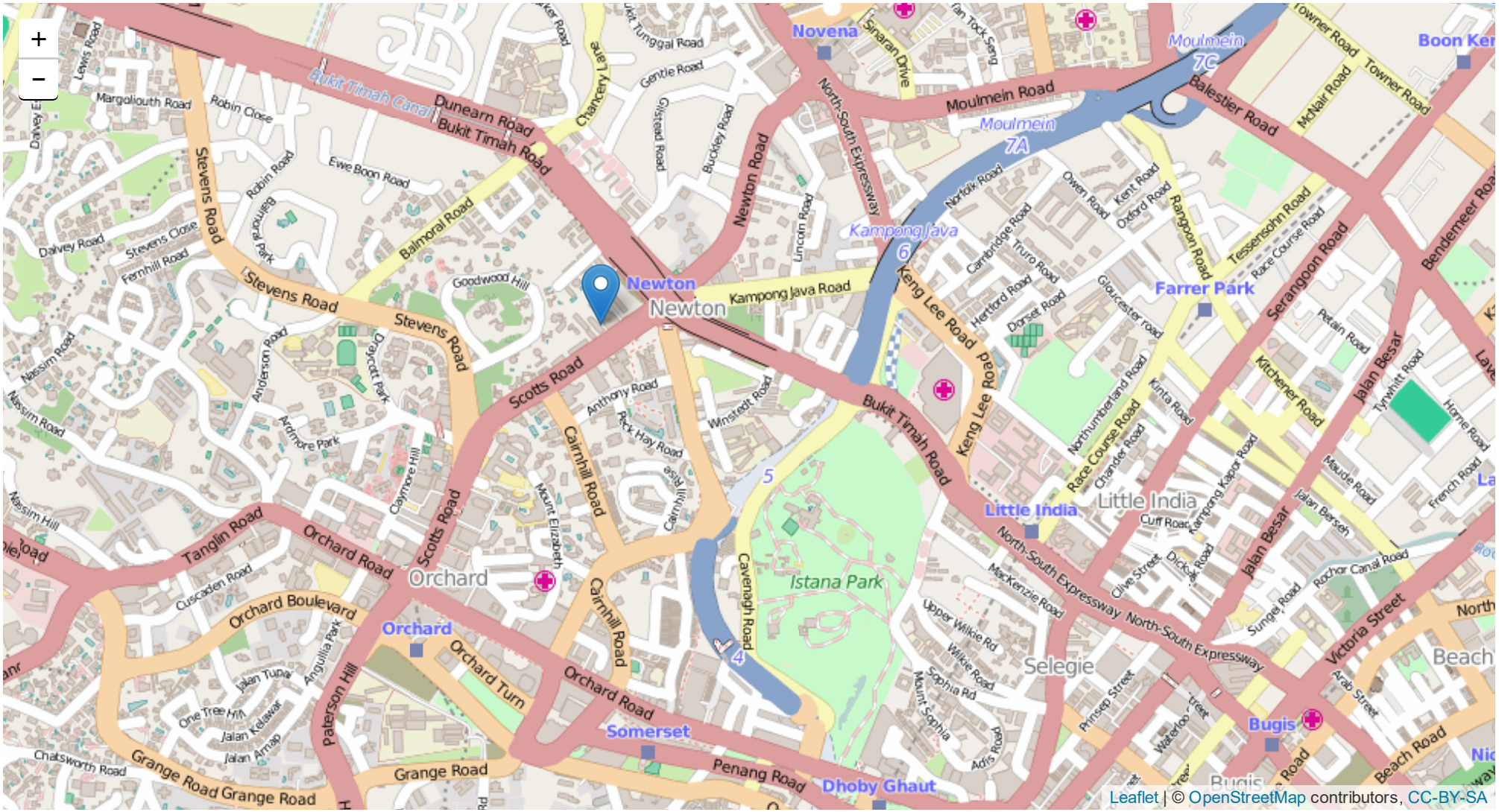
- Install leaflet via github if you haven't (requires `devtools` library)

```
devtools::install_github('rstudio/leaflet')
```

- Call the library and initialize a leaflet object

```
require(leaflet)
m <- leaflet() %>%  # The dataset in a data.frame can be loaded as a parameter in the function
    # so that objects can be added in later more conveniently
    addTiles() %>%  # Adds the map tiles
    addMarkers(lng = 103.8371408, lat = 1.3123813,
               popup = "Nielsen Singapore!")  # Additional objects
m
```

LEAFLET "HELLO WORLD"



INSERTING MULTIPLE MARKERS

- Leaflet automatically looks for column headers that look like longitude and latitude

```
x <- data.frame(lon = c(103.836, 103.837, 103.838),
                lat = c(1.311, 1.311, 1.313),
                label = c("I", "Love", "SG"),
                color = c("red", "blue", "black"))

require(leaflet)
m <- leaflet(x) %>%
    addTiles() %>%
    addMarkers(popup = ~label)  # Note that lat and lng are not specified here.
# Notice also the use of "~"
m
```

INSERTING MULTIPLE MARKERS

- Specifying the column names for lng and lat

```
x <- data.frame(lon = c(103.836, 103.837, 103.838),
                lat = c(1.311, 1.311, 1.313),
                label = c("I", "Love", "SG"),
                color = c("red", "blue", "black"))

require(leaflet)
m <- leaflet(x) %>%
    addTiles() %>%
    addMarkers(lng = ~lon, lat = ~lat, popup = ~label)
m
```

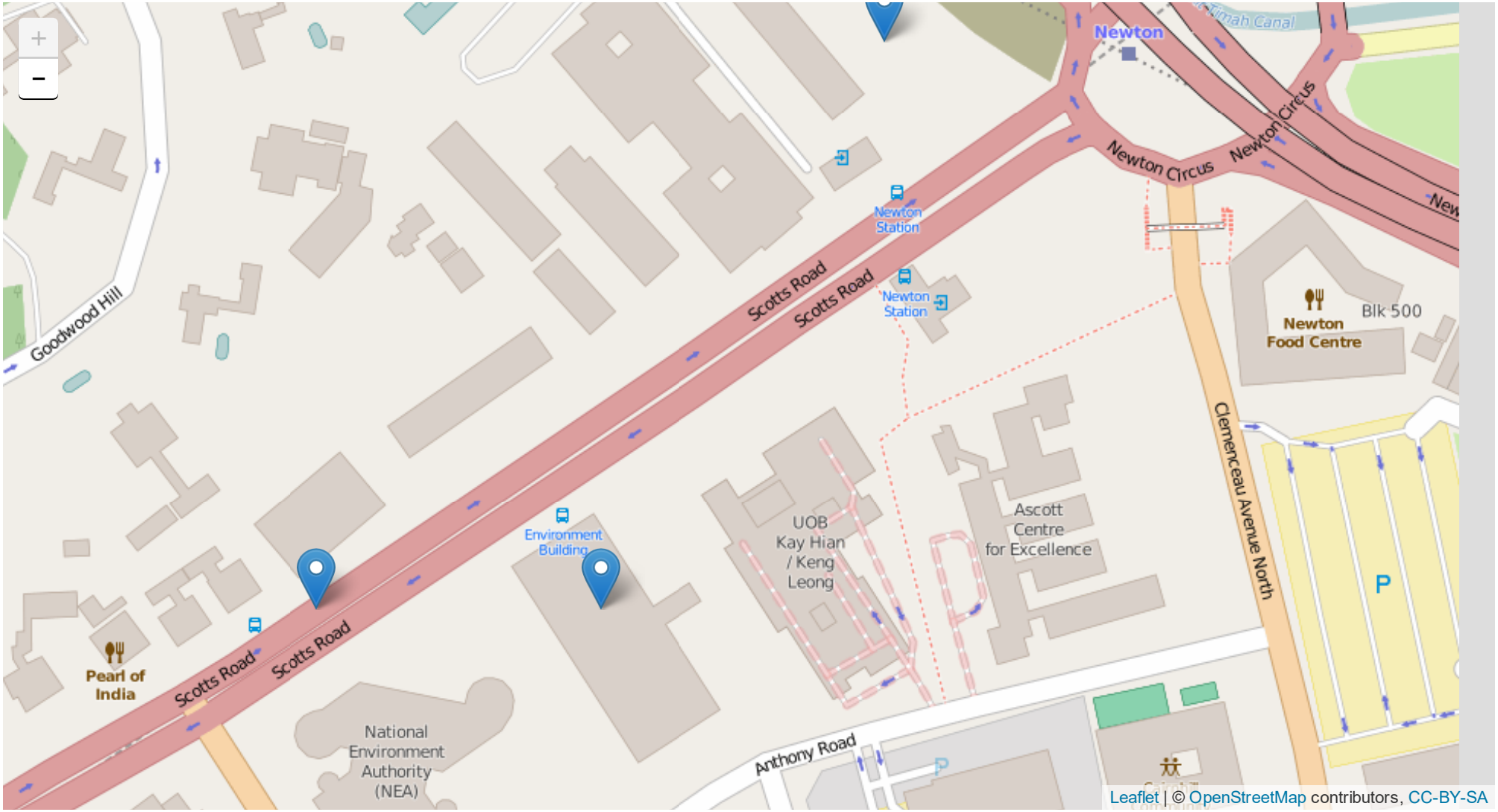
INSERTING MULTIPLE MARKERS

- An alternative method by calling the data.frame in the AddMarkers command

```
x <- data.frame(lon = c(103.836, 103.837, 103.838),
                lat = c(1.311, 1.311, 1.313),
                label = c("I", "Love", "SG"),
                color = c("red", "blue", "black"))

require(leaflet)
m <- leaflet() %>%
    addTiles() %>%
    addMarkers(data = x, popup = ~label)
m
```

INSERTING MULTIPLE MARKERS



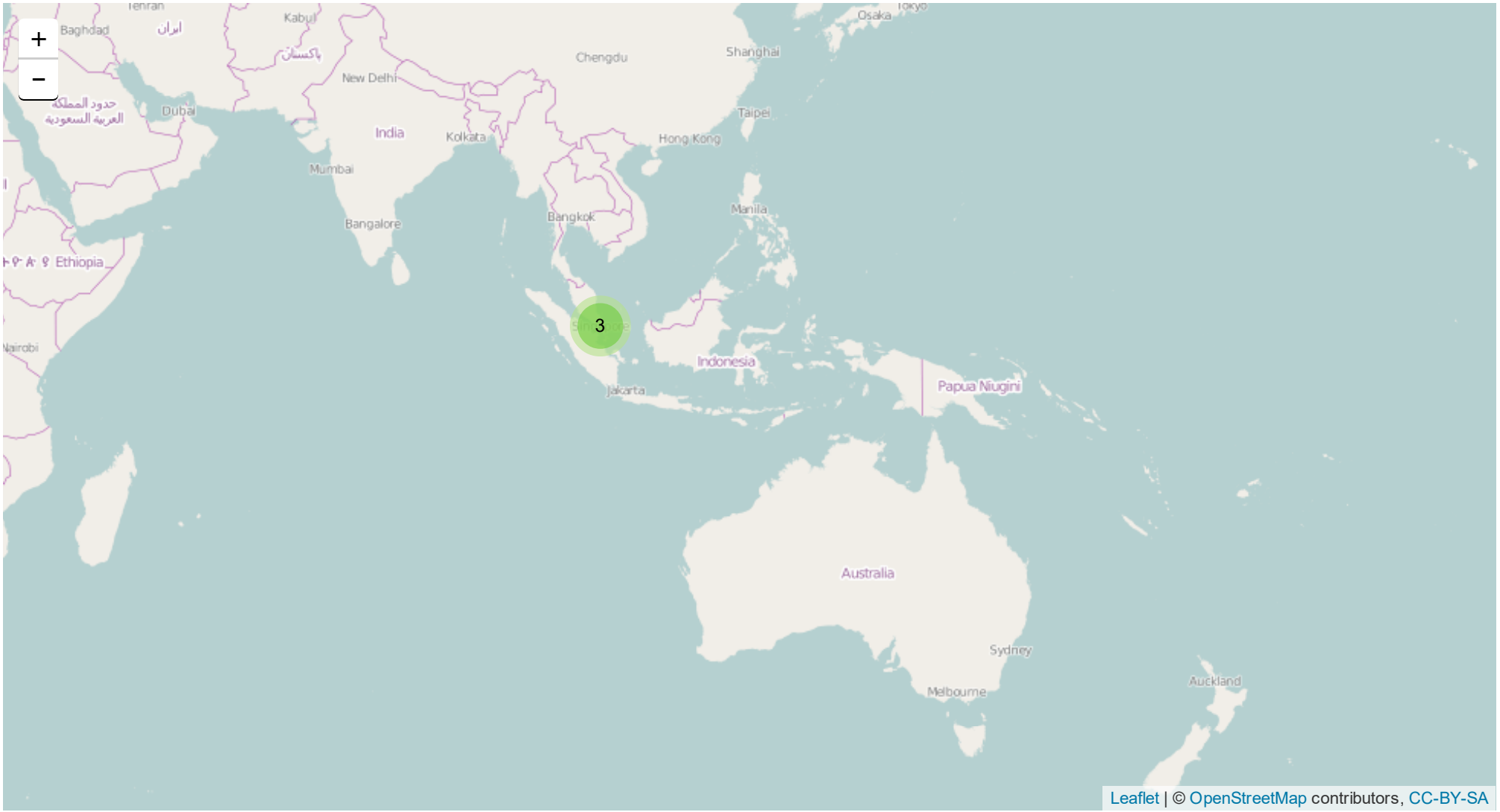
INSERTING MULTIPLE MARKERS

- Dynamic clustering of multiple markers

```
x <- data.frame(lon = c(103.836, 103.837, 103.838),
                lat = c(1.311, 1.311, 1.313),
                label = c("I", "Love", "SG"),
                color = c("red", "blue", "black"))

require(leaflet)
m <- leaflet(x) %>%
    addTiles() %>%
    setView(lng = mean(x$lon), lat = mean(x$lat), zoom = 3) %>%
    addMarkers(popup = ~label,
               clusterOptions = markerClusterOptions())  # Cluster the markers
m
```

INSERTING MULTIPLE MARKERS



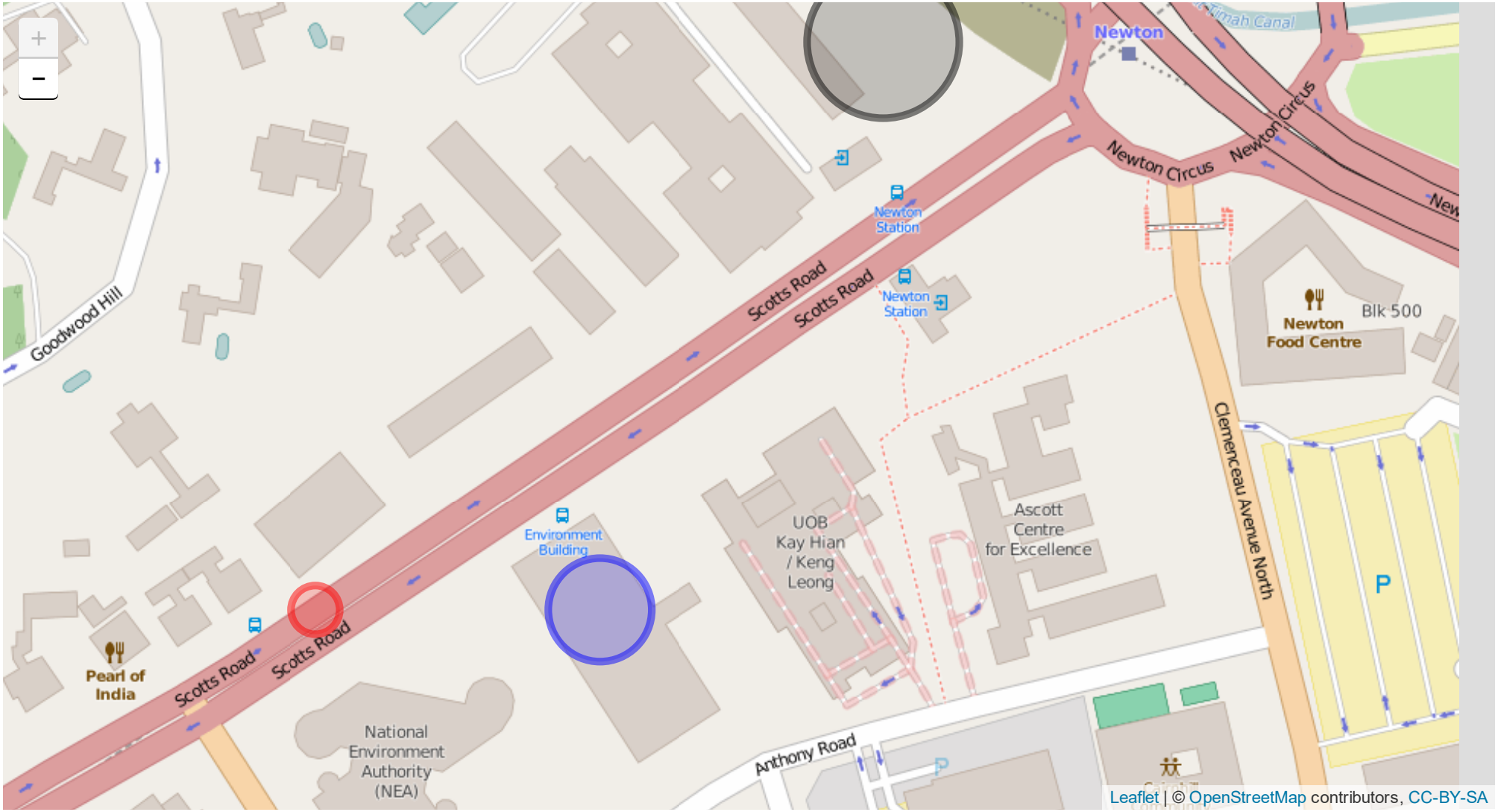
PLOTTING CIRCLES

- Circles allow for more customization in terms of colour and size

```
x <- data.frame(lon = c(103.836, 103.837, 103.838),
                lat = c(1.311, 1.311, 1.313),
                label = c("I", "Love", "SG"),
                color = c("red", "blue", "black"))

require(leaflet)
m <- leaflet(x) %>%
    addTiles() %>%
    addCircles(popup = ~label, color = ~color, radius = c(10, 20, 30))
m
```

PLOTTING CIRCLES



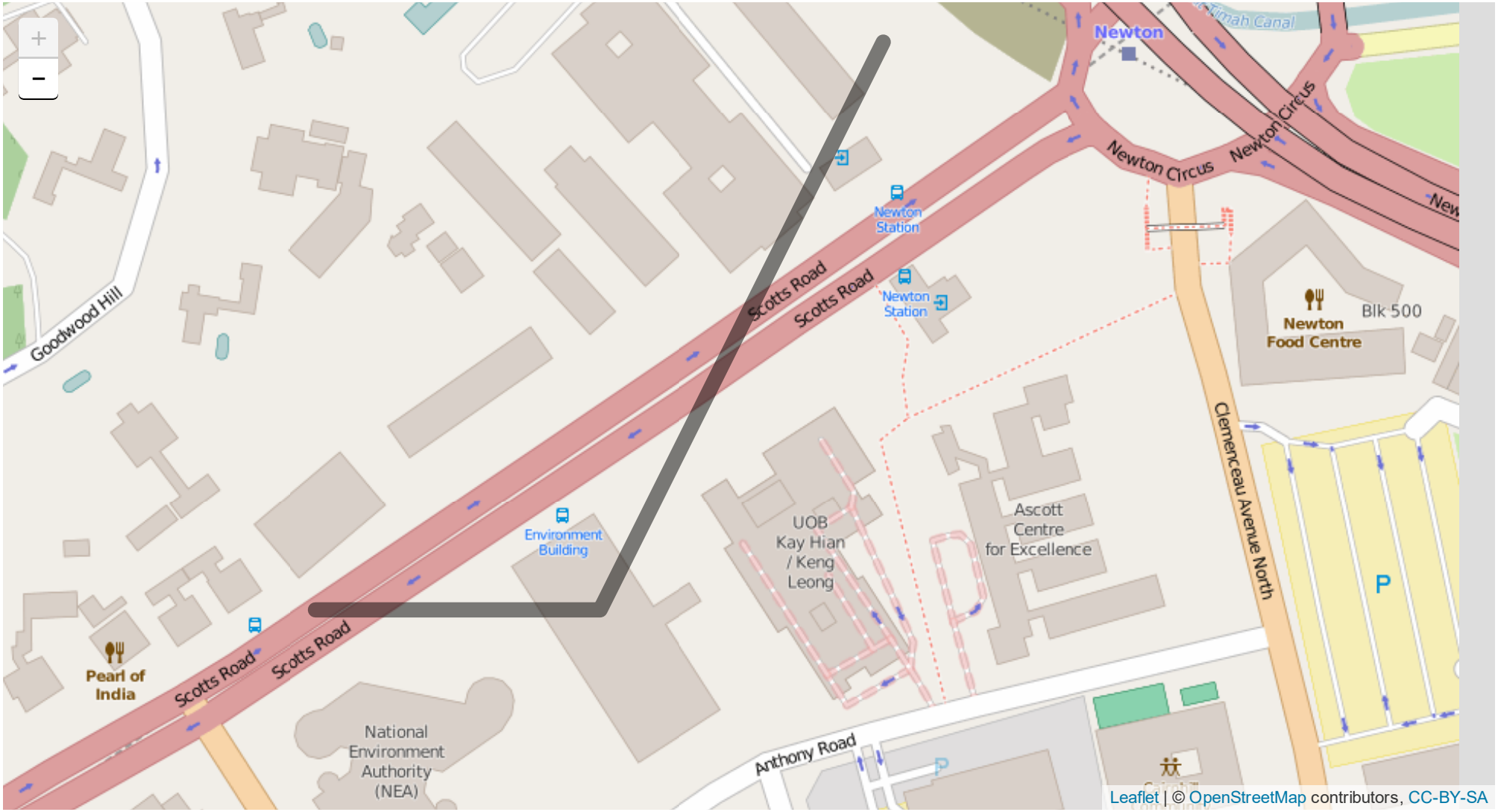
PLOTTING POLYLINES

- Creating paths using coordinates are similar, but `lng` and `lat` need to be specified

```
x <- data.frame(lon = c(103.836, 103.837, 103.838),
                lat = c(1.311, 1.311, 1.313),
                label = c("I", "Love", "SG"),
                color = c("red", "blue", "black"))

require(leaflet)
m <- leaflet(x) %>%
    addTiles() %>%
    addPolylines(lng = ~lon, lat = ~lat,
                 popup = "My Path", weight = 10, color = "black")
m
```

PLOTTING POLYLINES



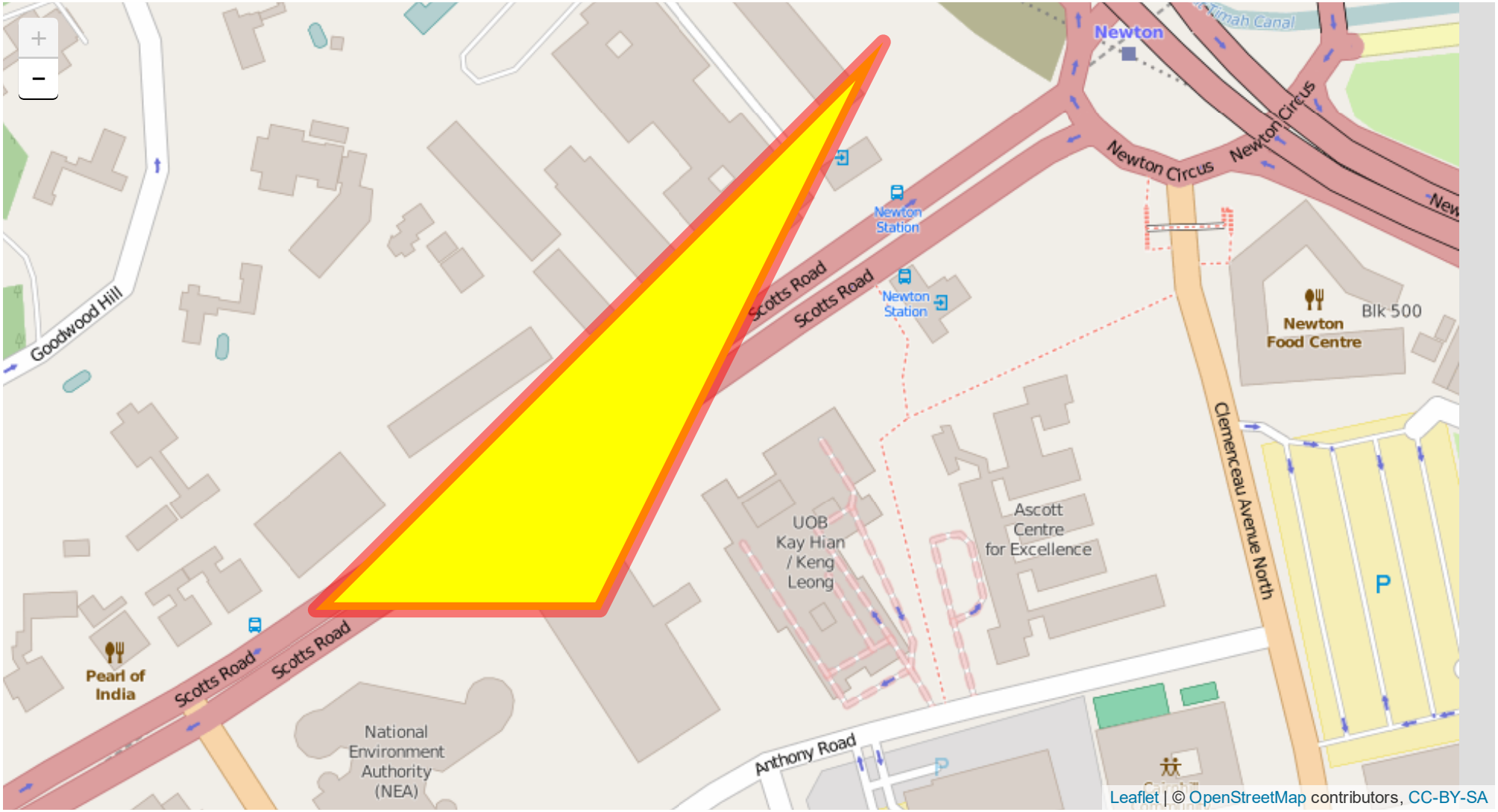
PLOTTING POLYGONS

- Leaflet completes the polygon automatically
- To plot polygons with holes, need to use polygon objects instead

```
x <- data.frame(lon = c(103.836, 103.837, 103.838),
                lat = c(1.311, 1.311, 1.313),
                label = c("I", "Love", "SG"),
                color = c("red", "blue", "black"))

require(leaflet)
m <- leaflet(x) %>%
    addTiles() %>%
    addPolygons(lng = ~lon, lat = ~lat,
                popup = "My Polygon", weight = 10, color = "red",
                fillColor = "", fillOpacity = 1)
m
```

PLOTTING POLYGONS



HANDLING SHAPE FILES

- Install `rgdal` package from CRAN: `install.packages("rgdal")`
- Download the DENGUE_CLUSTER zip file and unzip it into your working directory

```
require(rgdal)
# Reads the Shapefile into R
dengue <- readOGR(dsn = ".", layer = "DENGUE_CLUSTER", verbose = F)
# dsn: data source name (folder path)
# layer: shapefile name
# verbose option: turns progress report on or off
class(dengue)
```

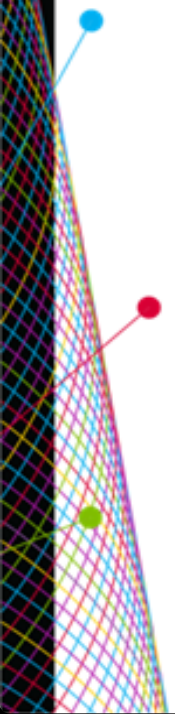
```
[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"
```

HANDLING SHAPE FILES

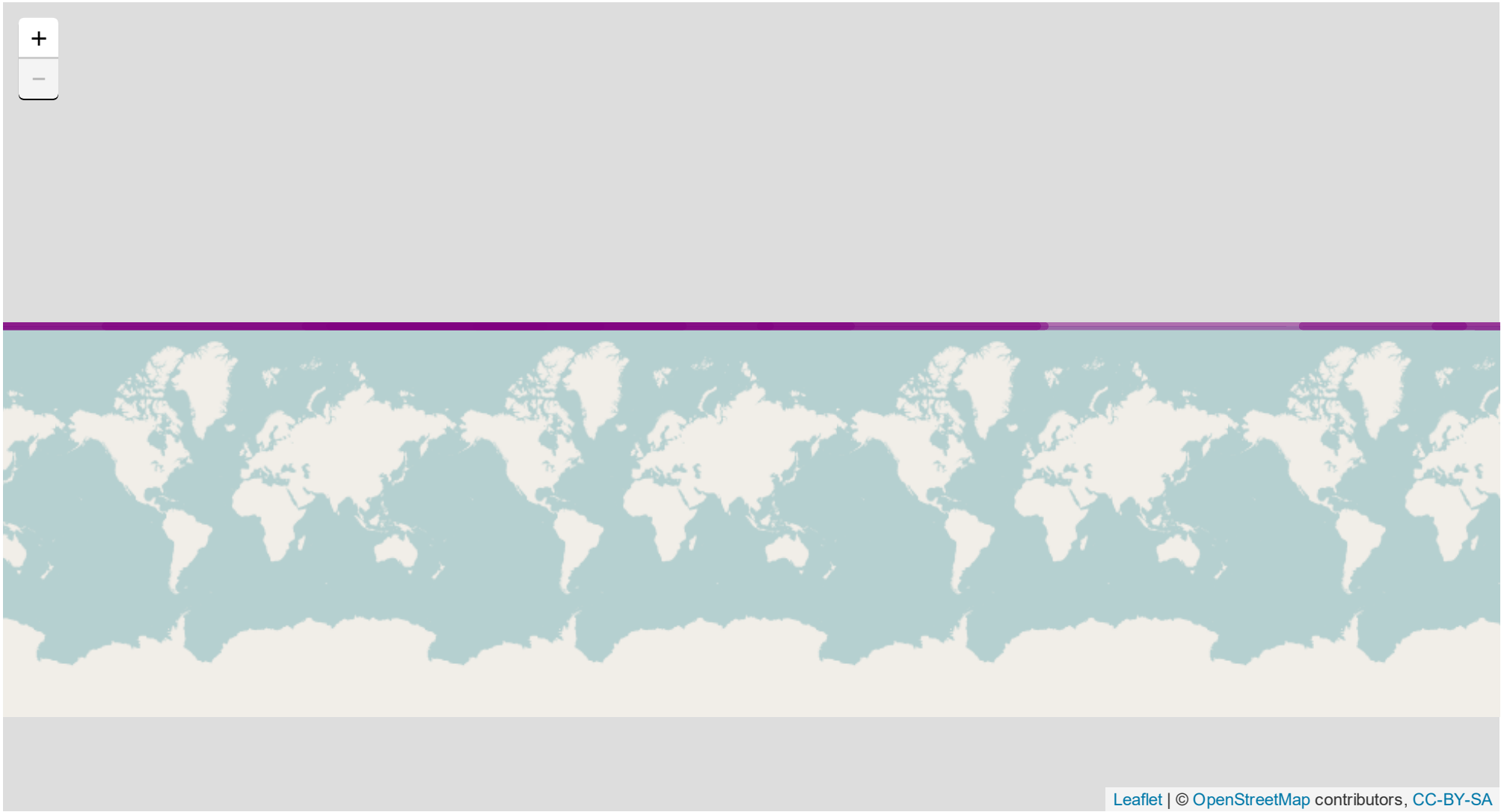
```
plot(dengue, col = "purple")
```

HANDLING SHAPE FILES

```
require(rgdal); require(leaflet)
dengue <- readOGR(dsn = ".", layer = "DENGUE_CLUSTER", verbose = F)
m <- leaflet(dengue) %>%
    addTiles() %>%
    addPolygons(color = "purple")
m
```



HANDLING SHAPE FILES



HANDLING SHAPE FILES

- Exploring the Spatial Polygon object

```
require(rgdal)
dengue <- readOGR(dsn = ".", layer = "DENGUE_CLUSTER", verbose = F)
dengue@proj4string  # Not in the "default" system in leaflet!
```

```
CRS arguments:
 +proj=tmerc +lat_0=1.366666666666667 +lon_0=103.8333333333333 +k=1
+x_0=28001.642 +y_0=38744.572 +datum=WGS84 +units=m +no_defs
+ellps=WGS84 +towgs84=0,0,0
```

HANDLING SHAPE FILES

- Exploring the Spatial Polygon object

```
head(dengue@data)
```

```
'data.frame':   55 obs. of  11 variables:
 $ OBJECTID  : int  1 2 3 4 5 6 7 8 9 10 ...
 $ LOCALITY  : Factor w/ 55 levels "Ang Mo Kio Ave 10 (Blk 438, 441, 444, 467)",..: 8 53 55 54 5 39 12
 $ CASE_SIZE : int  20 2 2 5 27 3 3 5 2 2 ...
 $ NAME      : Factor w/ 1 level "Dengue_Cluster": 1 1 1 1 1 1 1 1 1 1 ...
 $ HYPERLINK : Factor w/ 55 levels "http://www.dengue.gov.sg/cms/ehd/20150835.jpg",..: 2 33 41 12 3 32
 $ INC_CRC   : Factor w/ 55 levels "0270082AB4E215A9",..: 45 47 3 14 30 5 43 8 26 15 ...
 $ FMEL_UPD_D: Factor w/ 8 levels "2015/09/29","2015/09/30",..: 6 6 6 1 7 7 7 7 7 7 ...
 $ X_ADDR    : num  27974 27823 28176 28465 40743 ...
 $ Y_ADDR    : num  37668 45909 43829 45856 33557 ...
 $ SHAPE_Leng: num  3657 1536 1440 2193 3436 ...
 $ SHAPE_Area: num  384098 92557 84050 219705 312069 ...
```

HANDLING SHAPE FILES

Useful functions in `rgdal` package:

- `readOGR`: reads shapefile into R
- `CRS`: sets a coordinate reference system
- `coordinates`: assigns column headers of a data.frame as coordinates
- `proj4string`: sets the CRS of a Spatial Polygon object
- `spTransform`: transforms a data from one CRS to another

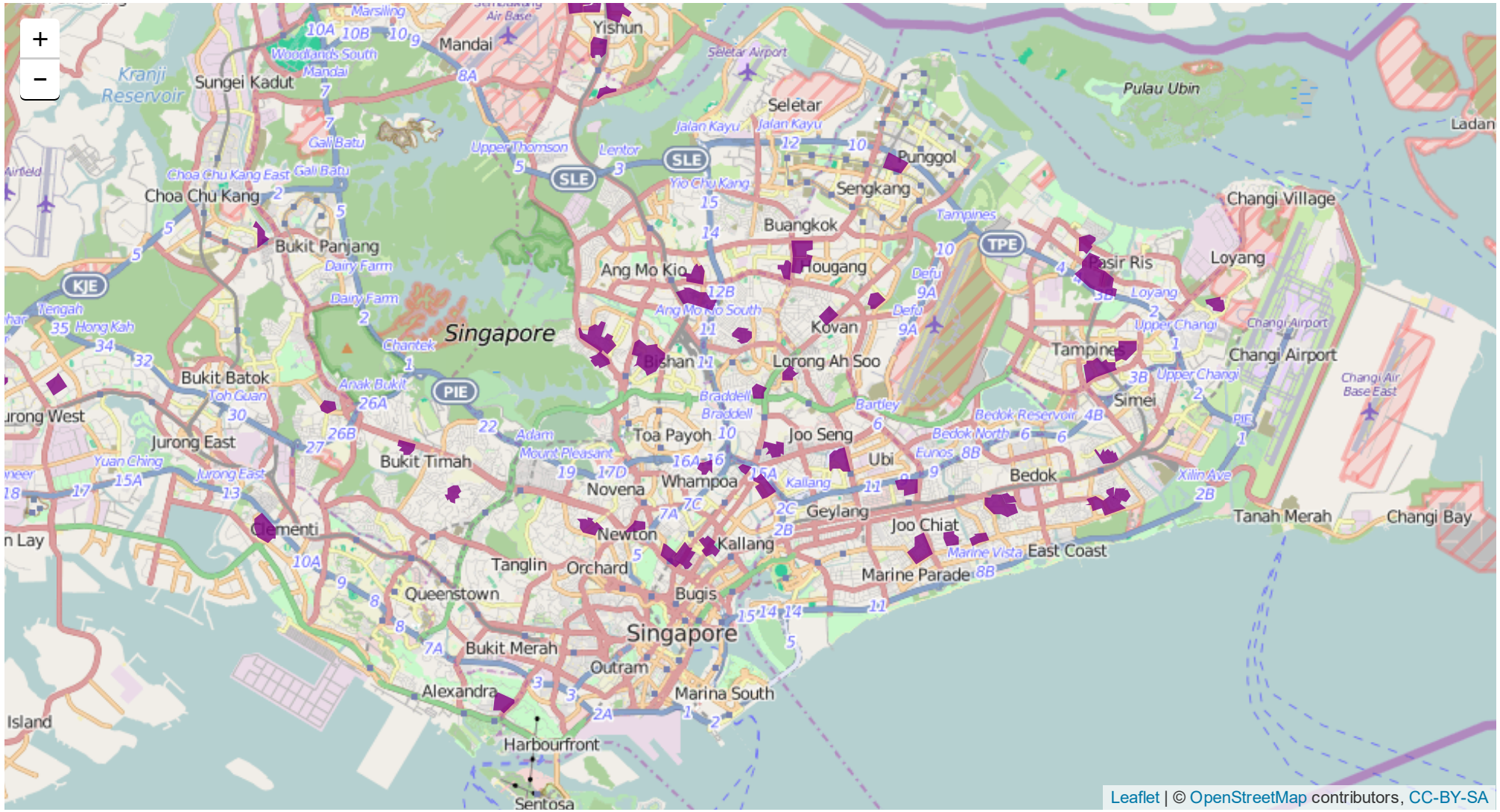
HANDLING SHAPE FILES: ADDING POLYGONS

```
require(rgdal); require(leaflet)
dengue <- readOGR(dsn = ".", layer = "DENGUE_CLUSTER", verbose = F)

dengue.new <- spTransform(dengue, CRS("+init=epsg:4326"))  # Converts to the usual CRS

m <- leaflet(dengue.new) %>% addTiles() %>%
    addPolygons(color = "purple",
                stroke = F,
                fillOpacity = 0.8,
                popup = as.character(dengue@data$CASE_SIZE))  # popups only take in characters!
m
```

HANDLING SHAPE FILES: ADDING POLYGONS



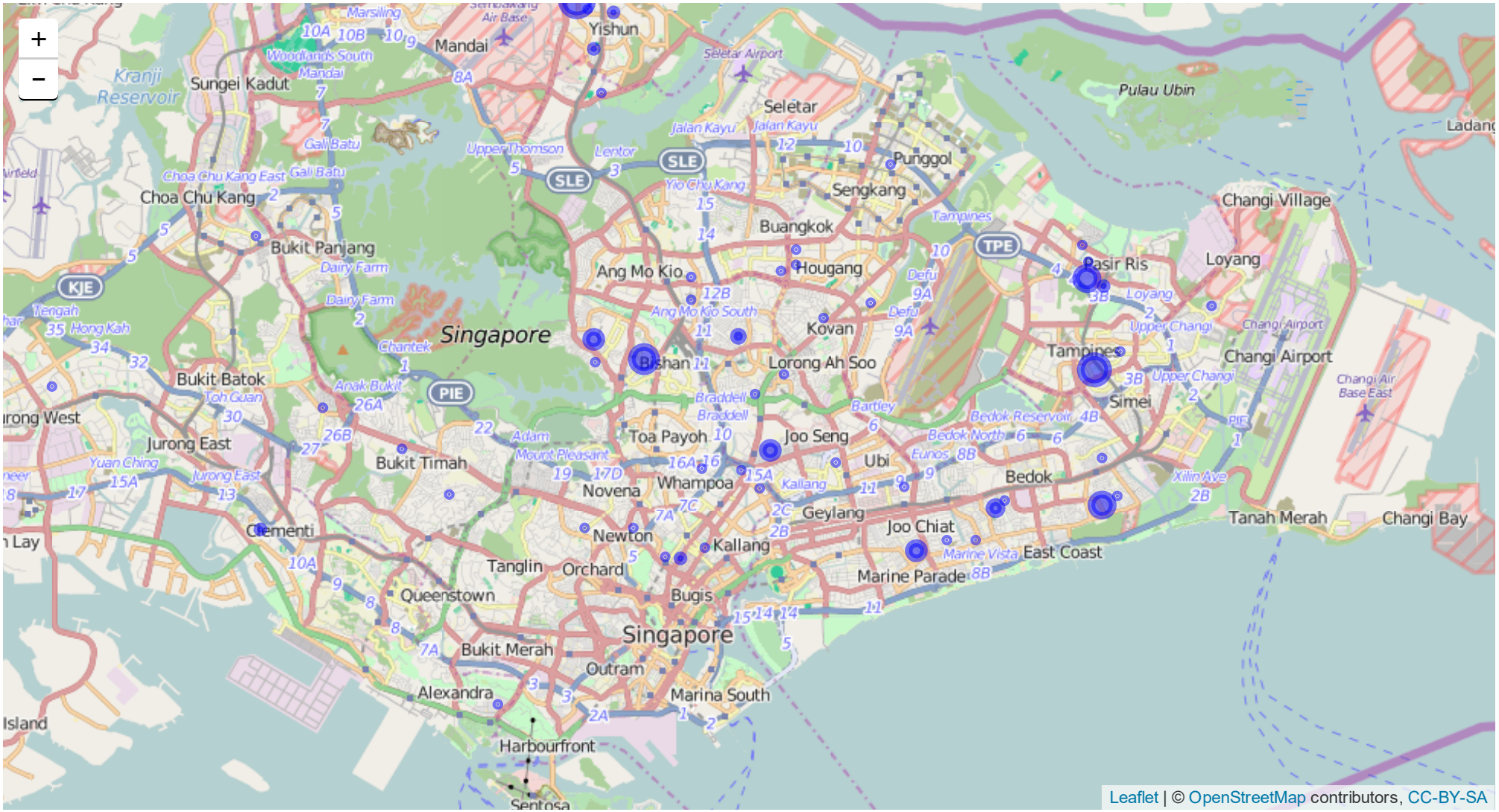
USING THE DATA FILE: ADDING CIRCLES

```
require(rgdal); require(leaflet)
dengue <- readOGR(dsn = ".", layer = "DENGUE_CLUSTER", verbose = F)

# Make use of the data to obtain coordinates for circles
dengue.circles <- dengue@data
coordinates(dengue.circles) <- c("X_ADDR", "Y_ADDR")
proj4string(dengue.circles) <- dengue@proj4string
dengue.circles <- spTransform(dengue.circles, CRS("+init=epsg:4326"))

m <- leaflet(dengue.circles) %>% addTiles() %>%
    addCircles(radius = dengue@data$CASE_SIZE*10,
               color = "blue",
               opacity = 0.5,
               fillOpacity = 0.5,
               popup = as.character(dengue@data$CASE_SIZE))
m
```

USING THE DATA FILE: ADDING CIRCLES

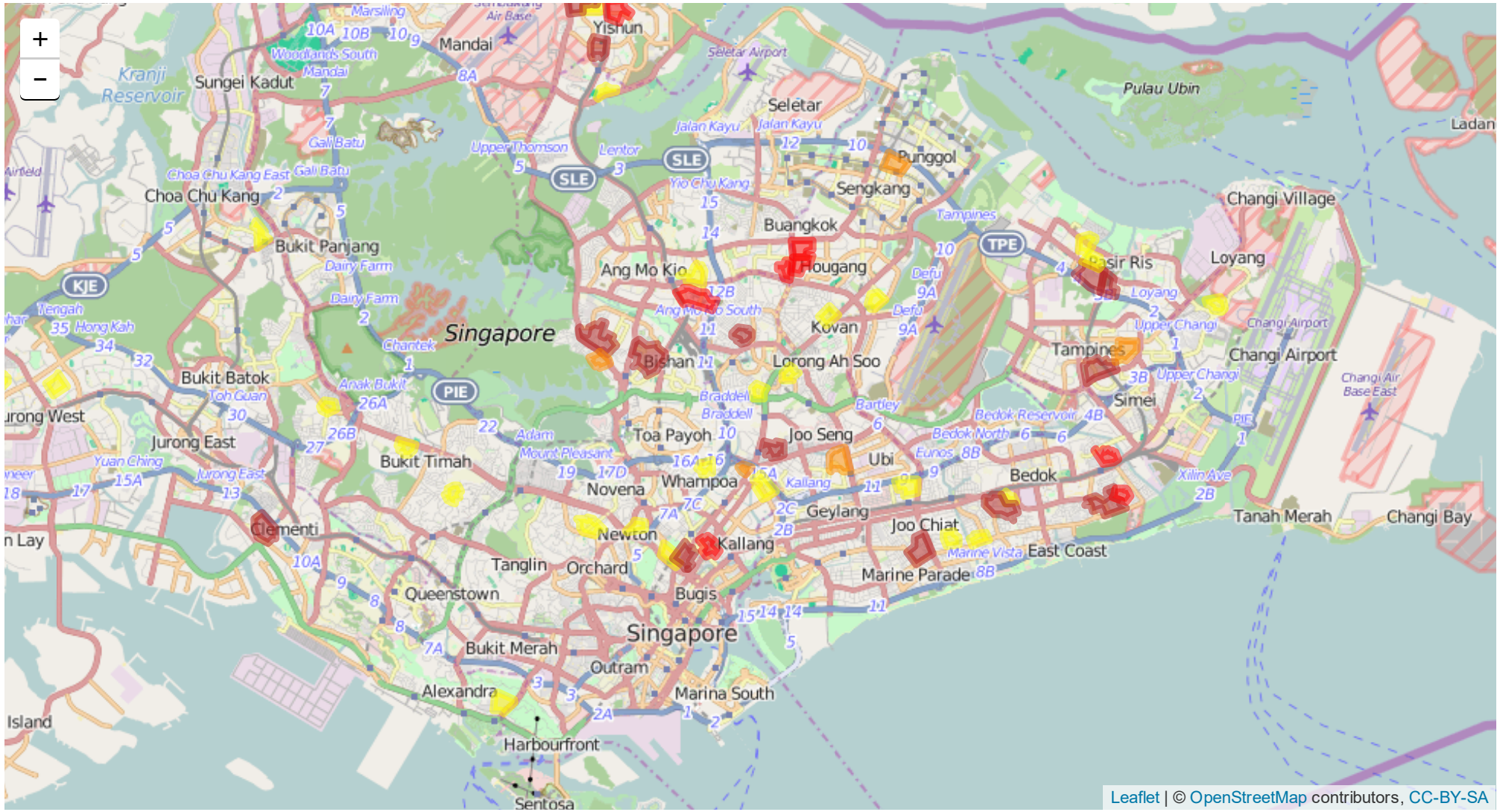


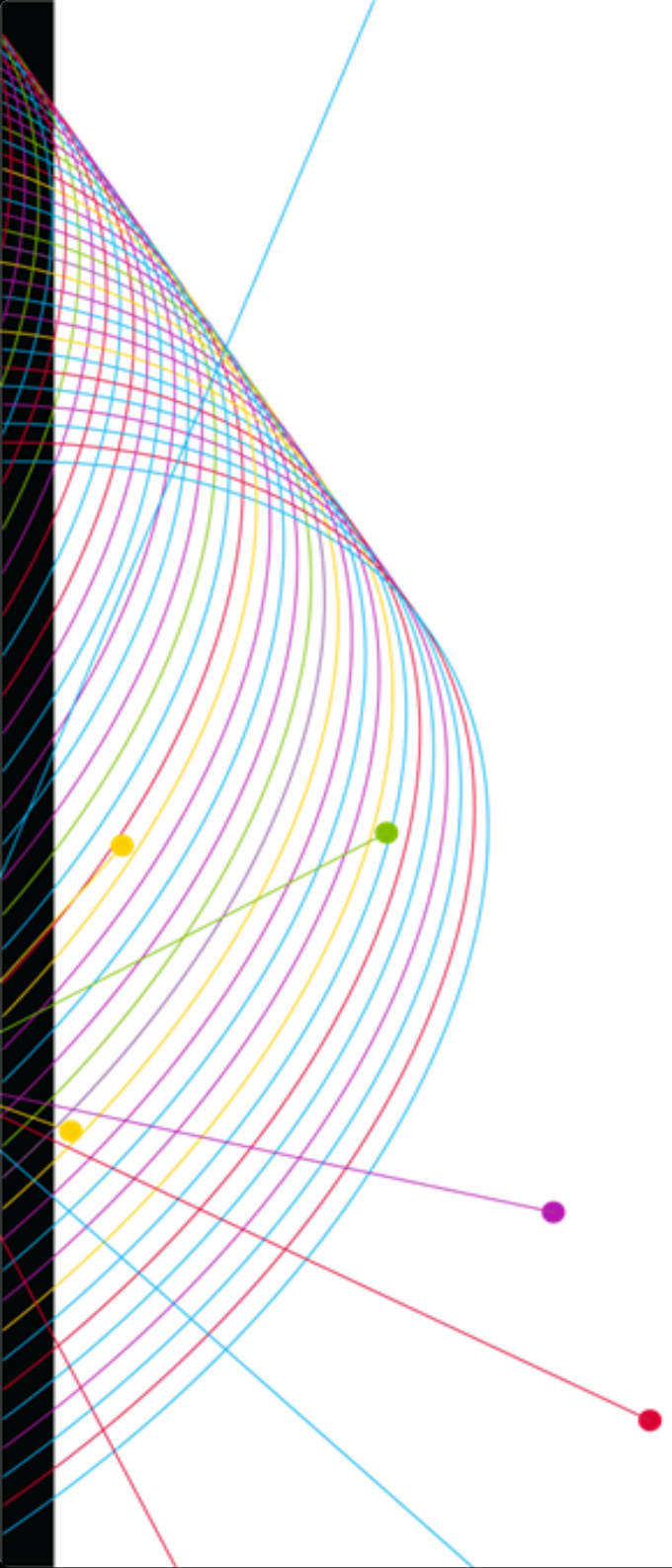
PUTTING THEM TOGETHER

```
require(rgdal); require(leaflet)
dengue <- readOGR(dsn = ".", layer = "DENGUE_CLUSTER", verbose = F)
dengue.new <- spTransform(dengue, CRS("+init=epsg:4326"))

# Create color vector
dengue.color <- ifelse(dengue@data$CASE_SIZE >
                           quantile(dengue@data$CASE_SIZE, probs = 0.75), "firebrick",
                       ifelse(dengue@data$CASE_SIZE >
                                  quantile(dengue@data$CASE_SIZE, probs = 0.50), "red",
                              ifelse(dengue@data$CASE_SIZE >
                                         quantile(dengue@data$CASE_SIZE, probs = 0.25), "darkorange",
                                     "yellow")))
m <- leaflet(dengue.new) %>% addTiles() %>%
    addPolygons(fillOpacity = 0.5, color = dengue.color,
                popup = as.character(dengue@data$CASE_SIZE))
m
```

PUTTING THEM TOGETHER





Simple Spatial Analysis in R

Simple Spatial Analysis in R

Simple Spatial Analysis in R

Common libraries for spatial analysis:

- `rgeos` (methods for topological data)
- `sp` (methods for spatial data)
- `raster` (methods for raster data)
- `geosphere` (methods for unprojected data)

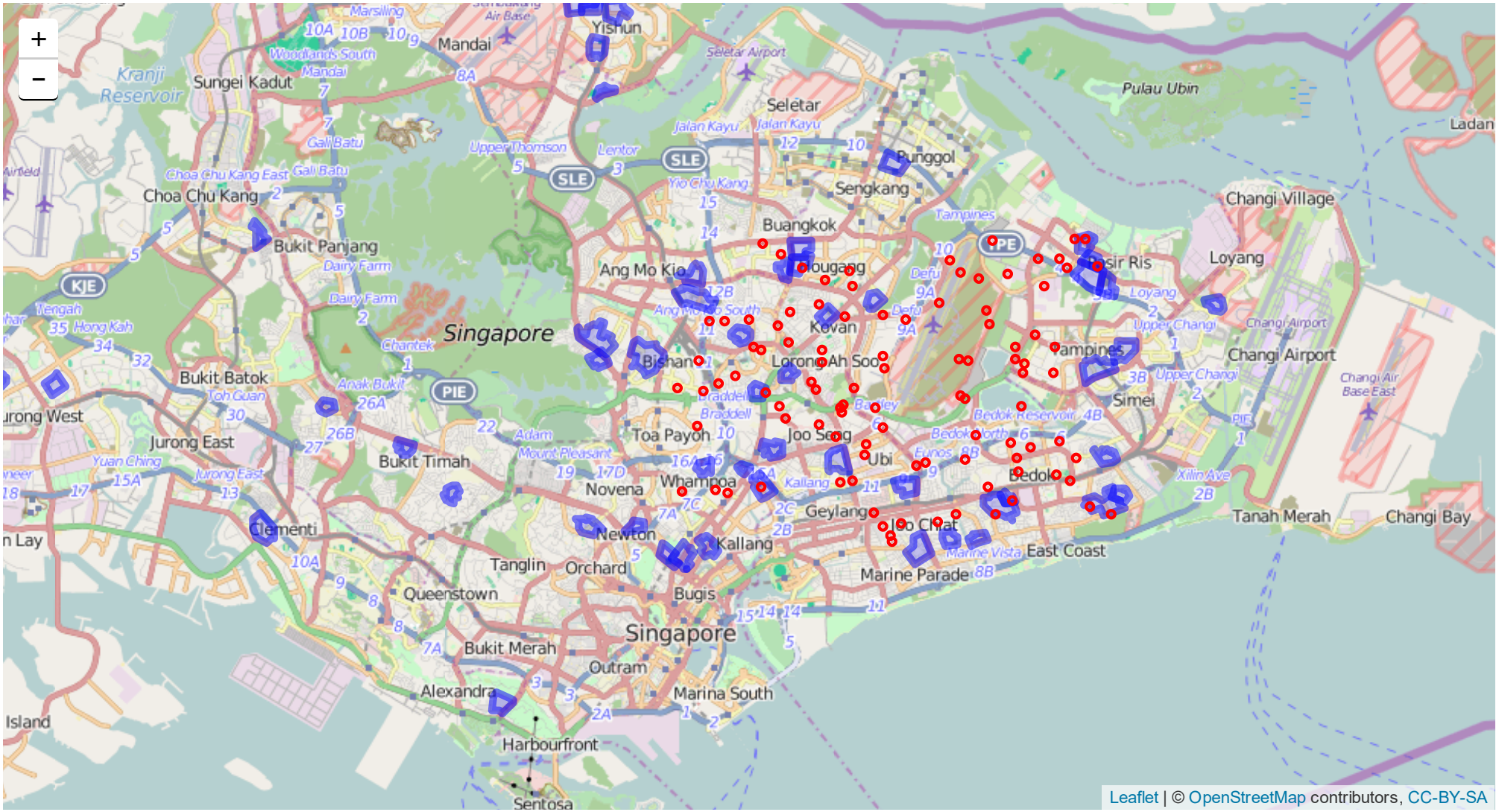
Simple Spatial Analysis in R: Counting Overlaps

```
require(rgdal); require(leaflet)
dengue <- readOGR(dsn = ".", layer = "DENGUE_CLUSTER", verbose = F)
dengue.new <- spTransform(dengue, CRS("+init=epsg:4326"))

# Create random points and set them with the usual CRS in order to count overlaps
set.seed(1)
to.count <- data.frame(lon = runif(n = 100, min = 103.85, max = 103.95),
                       lat = runif(n = 100, min = 1.31, max = 1.38),
                       sn = 1:100)
coordinates(to.count) <- c("lon", "lat")
proj4string(to.count) <- CRS("+init=epsg:4326")

m <- leaflet() %>% addTiles() %>%
    addPolygons(data = dengue.new, color = "blue") %>%
    addCircles(data = to.count, color = "red", opacity = 1,
               popup = as.character(to.count@data$sn))
m
```

Simple Spatial Analysis in R: Counting Overlaps



Simple Spatial Analysis in R: Counting Overlaps

```
# Use the "over" function to obtain overlaps
count.overlaps <- over(dengue.new, to.count)
table(count.overlaps)
```

```
count.overlaps
 3 46 61 88 94
 1  1  1  1  1
```

```
sum(table(count.overlaps))
```

```
[1] 5
```

Simple Spatial Analysis in R: Creating Buffers

```
require(rgdal); require(rgeos); require(leaflet)
# Geo-coordinates for two polylines (coordinates are the "joins")
l1 <- data.frame(lon = c(103.836, 103.837, 103.838),
                 lat = c(1.311, 1.311, 1.312))
l2 <- data.frame(lon = c(103.835, 103.837, 103.838),
                 lat = c(1.310, 1.312, 1.313))

# Convert to polyline, and then to Spatial Lines with the usual CRS
sl <- SpatialLines(list(Lines(list(Line(l1)), ID = "line1"),
                        Lines(list(Line(l2)), ID = "line2")))
proj4string(sl) <- CRS("+init=epsg:4326")
sl.df <- SpatialLinesDataFrame(sl, data = data.frame(row.names = c("line1", "line2"),
                                                     index = c("line1", "line2"),
                                                     color = c("blue", "green")))
```

Simple Spatial Analysis in R: Creating Buffers

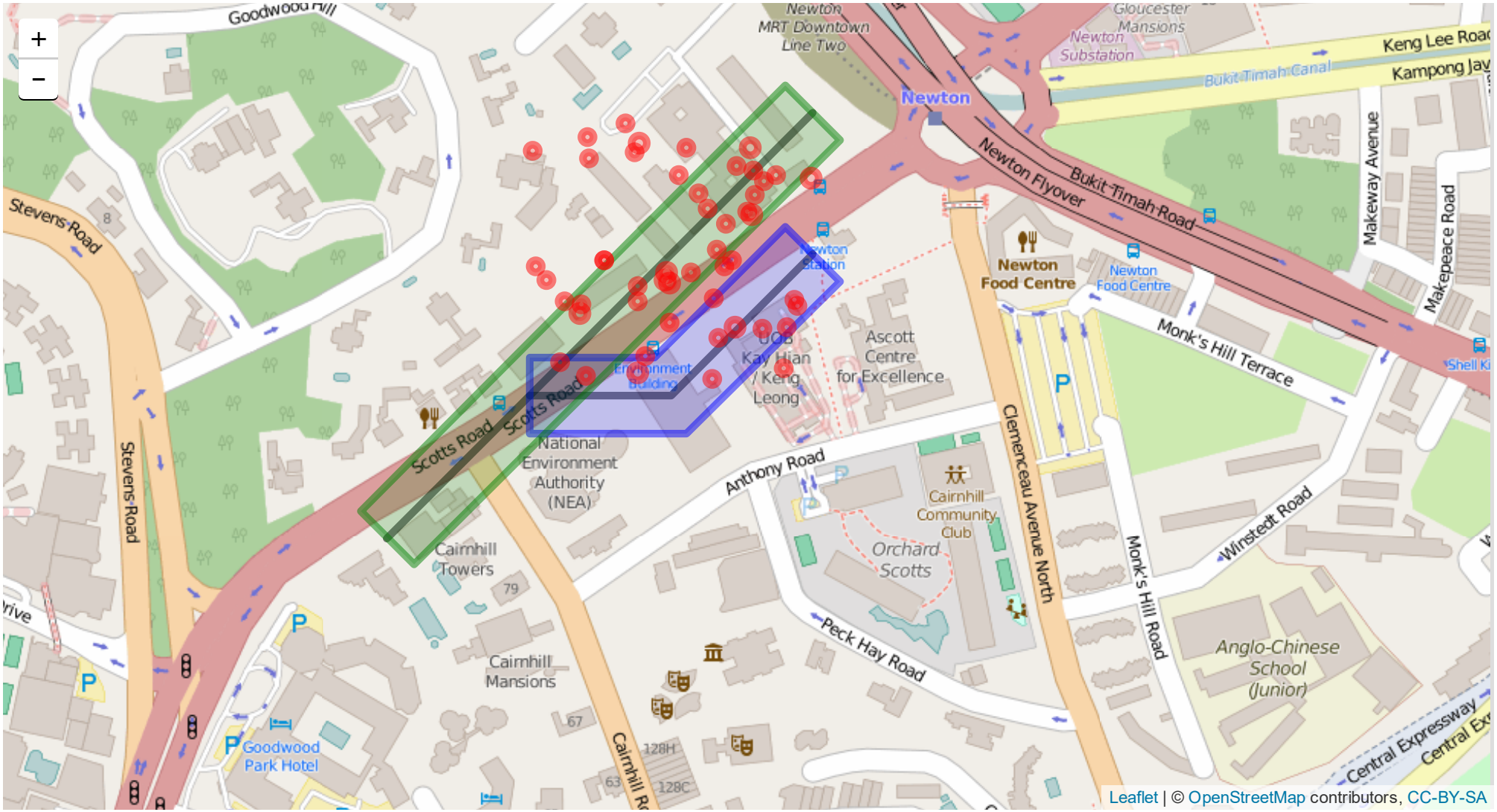
```
# Create a 30-meter buffer for the Spatial Line
sl.buffer <- spTransform(sl.df, CRS("+init=epsg:3857"))
sl.buffer <- gBuffer(sl.buffer, byid = T, width = 30, quadsegs = 5,
                     capStyle = "FLAT", joinStyle = "MITRE", mitreLimit = 3)
sl.buffer <- spTransform(sl.buffer, CRS("+init=epsg:4326"))

# Generate random points around the area with the usual CRS
set.seed(1)
random.points <- data.frame(lon = runif(50, min = 103.836, max = 103.838),
                            lat = runif(50, min = 1.311, max = 1.313))
coordinates(random.points) <- c("lon", "lat")
proj4string(random.points) <- CRS("+init=epsg:4326")
```

Simple Spatial Analysis in R: Creating Buffers

```
m <- leaflet() %>%
    addTiles() %>%
    addPolylines(data = sl.df, color = "black") %>%
    addPolygons(data = sl.buffer,
                popup = sl.buffer@data$index,
                color = sl.buffer@data$color) %>%
    addCircles(data = random.points, color = "red", radius = 5)
m
```

Simple Spatial Analysis in R: Creating Buffers



Simple Spatial Analysis in R: Creating Buffers

```
# Non-duplicated counts: considers sl.buffer as a single object
count.overlaps.nonduplicate <- over(random.points, sl.buffer)
table(count.overlaps.nonduplicate)
```

```
       color
index   blue green
  line1   10     0
  line2    0    20
```

```
# Duplicated counts: Each "line" in sl.buffer is a different object
count.overlaps.duplicate <- over(sl.buffer, random.points, returnList = T)
sapply(count.overlaps.duplicate, length)
```

```
line1 line2
   10    22
```

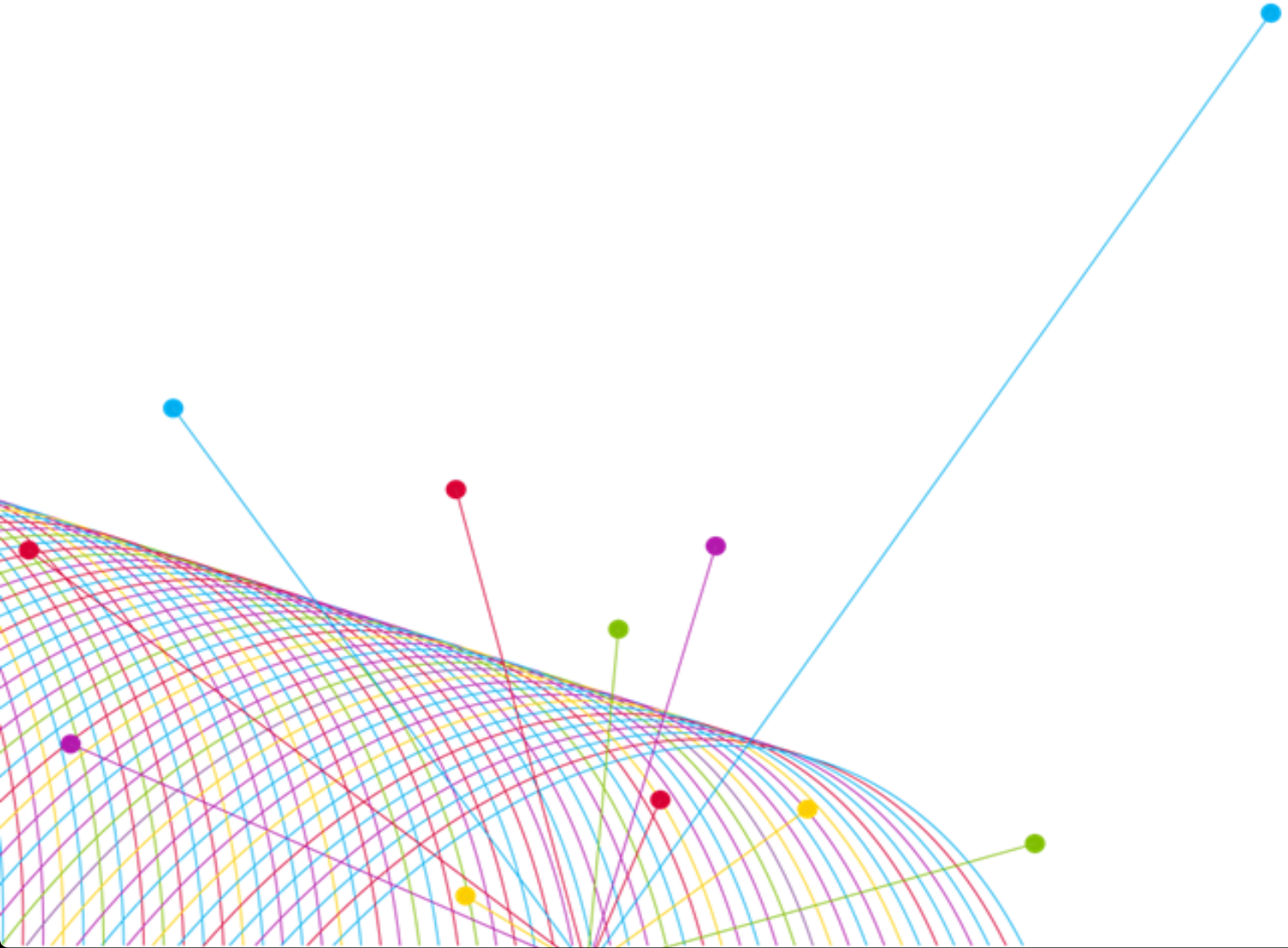
Some Useful Resources

- https://rstudio.github.io/leaflet
- http://rstudio-pubs-static.s3.amazonaws.com/7993_6b081819ba184047802a508a7f3187cb.html
- http://spatial.ly

nielsen

AN UNCOMMON SENSE OF THE CONSUMER™

Thank You!



/